SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 8, 79–100 (1996)

Practice

Defining and Implementing a Measurement-based Software Maintenance Process

JOEL HENRY
Department of Computer and Information Science, East Tennessee State University, Box 70711, Johnson City, TN, 37614, U.S.A.

ROBERT BLASEWITZ
Lockheed-Martin, Government Electronic Systems Division, Moorestown, NJ, 08057, U.S.A.

DAVID KETTINGER
Integrated Systems Consulting Group, 575 E. Swedesford Road, Suite 200, Wayne, PA, 19087, U.S.A.

SUMMARY

This paper describes the measurement-based software maintenance process defined and implemented at Lockheed-Martin, Moorestown, NJ. The documented process includes extensive data collection, a tightly controlled but highly accessible database, data analysis techniques supported by software tools, and process assessment and improvement activities. The methods and techniques used are presented in a 'how to' fashion so that other organizations can leverage our efforts to define and implement a measurement-based process of their own. Our approach is an evolutionary one, rather than a revolutionary organizational upheaval. We describe the benefits gained from our process, including statistically validated metric results, and the subsequent process improvements implemented. This paper describes solutions to the 'real-world' issues faced by an organization which successfully implemented a measurement-based software maintenance process.

KEY WORDS: software maintenance; software metrics; software assessment

1. INTRODUCTION

Software maintenance continues to be a major challenge in the software industry. Software engineers often avoid maintenance projects despite the fact that maintaining software extends over many years and generates more income than the original development effort. Project managers struggle to estimate maintenance effort, schedule and cost. Most experts agree that these problems result from the very different types of activities taking place during maintenance, the lack of useful design and code documentation, the inability to determine accurately the complete impact of change, and other important factors. Unfortunately, even today, exten-

CCC 1040–550X/96/020079–22

Received 10 February 1995
Revised 24 September 1995

sive quantitative assessment of the entire maintenance process has not been performed, and therefore validated quantitative models do not exist.

Recognizing solutions to these problems requires an organized, stepwise, long-term effort across the organization so Martin Marietta formed the Product Improvement Committee (PIC). The PIC consists of representatives from Martin Marietta Engineering, Program Management, Software Quality Assurance, the customer auditing agency (DPRO), software subcontractors (Computer Science Corporation) and Technical Representatives of the U.S. Navy (TECHREP). The PIC agreed to focus on software process improvement through process definition and software measurement. The specific goals of the PIC were:

- document the software process to ensure consistency across maintenance efforts
- implement extensive software process and product measurement
- track and trace product changes throughout the maintenance process
- permit process and product characteristics to be quantitatively evaluated and predicted
- support statistical analysis of process and product measures
- leverage the results to determine opportunities for process improvement

Achieving these goals required synergizing the work of the Software Engineering Institute (SEI), Management Science Techniques and Total Quality Management (TQM) principles. The SEI Capability Maturity Model provides an organized framework of software engineering goals and the practices needed to attain these goals. The field of Management Science consistently applies quantitative methods to control quality, assess risk, evaluate processes, and predict product and process characteristics. These methods applied to software maintenance provide solutions to quality, schedule, cost and co-ordination problems. TQM principles advocate teamwork, customer focus, defect prevention, and utilization of the talents of labour and management. The PIC applied methods and techniques from each of these approaches to process assessment and improvement.

This paper describes how these goals were accomplished, what the results were, and the lessons learned along the way. Specifically, Section 2 provides background on the organization, the product and the process. Section 3 describes the method used to model the maintenance process and includes an example. The methods and software tools used to measure and evaluate the process are presented in Section 4. Section 5 presents results and overall conclusions.

2. BACKGROUND

2.1. The software product

Martin Marietta serves as the maintenance contractor for a major radar-based weapons system. The weapons system enjoys a long history of success with the US Navy but the task of maintaining the software has become more complex because of multiple, concurrent maintenance efforts (including both corrective and perfective activities). Multiple organizations and multiple groups within organizations also participate in each maintenance effort making communication complex and co-ordination difficult. Control of computer programs and documentation, including specification changes, defect reports and error correction status can become overwhelming for multi-million source lines of code. Maintenance efforts operate

under fixed schedules, placing a great deal of pressure on the organizations, groups and individuals. The growing complexity of the software product and the maintenance effort requires a well-documented, measurement-based, continuously improving process in order to produce a high quality product on time and within shrinking budgets. A major accomplishment of the Martin Marietta maintenance team is that they have never missed a delivery schedule.

The large radar-based weapons system is controlled by 13 different computer programs executing on separate standard Navy hardware platforms. The system receives input from external sensors and operators. Extensive communication between the programs via a high-speed network takes place during operation. The weapons system has undergone extensive maintenance during its 23 year lifetime. The majority of maintenance efforts incorporate new functionality although existing errors are also corrected. Enhancing this functionality is required to keep abreast of new technological developments and potential threats to the ship's defence.

2.2. The existing maintenance process

The computer programs comprising the radar-based weapons system are released in baselines. A baseline is implemented under a contract which specifies a set of functional upgrades. Each upgrade is referred to as an upgrade item. Upgrade items typically require changes to the Program Performance Specification (PPS), the Interface Design Specification (IDS) or both. An overview of the process used to create new baselines is shown in Figure 1.

System engineers analyse the impact and interpret the precise meaning of each upgrade item during the system specification and software specification phases, depicted together as a Requirements Specification in Figure 1. This analysis is documented in the new PPS and the new IDS, which are reviewed at the Preliminary Design Review (PDR). The new PPS and new IDS are the basis for implementing the new baseline. Hereafter we will simply say PPS and IDS to refer to the new versions of these documents.

Detailed design of upgrade items takes place following the PDR. Detailed design of the functionality documented in the PPS and IDS typically require additions and changes to the PPS and the IDS. The IDS and PPS pages requiring change and the detailed design are

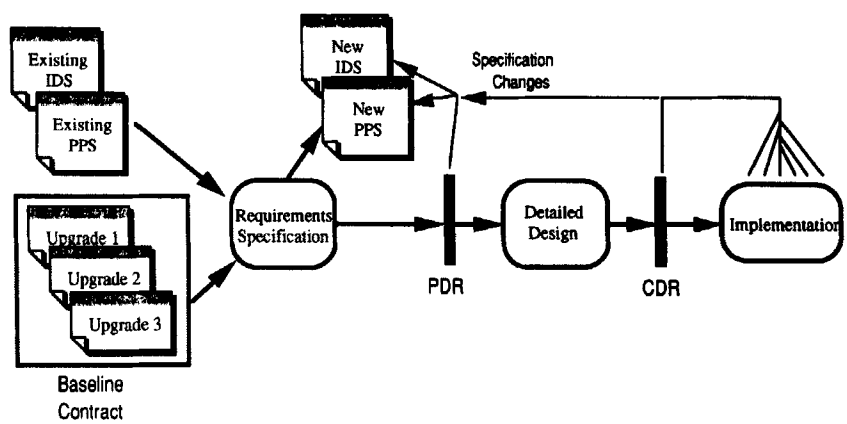

Figure 1. Baseline development process

reviewed at the Critical Design Review (CDR). Successful completion of the CDR, as judged by the customer representatives and management, initiates the implementation phase for changes to the computer programs.

The new versions of the PPS and the IDS presented at the PDR and the CDR are modified at the PDR, the CDR and post-CDR. These specification changes are documented using a specification change form and implemented following the CDR.

While the process described above appears well-defined, in fact implementation varied considerably over time and across baselines. Tracking and tracing of upgrade items, specification changes, defects and error corrections was difficult and error-prone. Few software measures were collected and little, if any, analysis of the measures was performed.

A documented software process must be produced before meaningful measurements can be specified and obtained. An accurate software process specification demonstrates a thorough understanding of the process, and illustrates that the process is indeed repeatable, which is required prior to selection of software measures. The software process document is used to define measurements and specify where in the software process the measures are collected.

3. MODELLING THE SOFTWARE PROCESS

3.1. The role of the software process

Software products, including both executable code and supporting documentation, are produced by the implementation of a software process. The software process consists of the activities, tools and methods used to produce software products (Humphrey, 1989). In order to understand, measure, and improve a software process, the process must be accurately documented in the form of a process model.

Process modelling, including both graphic and written descriptions, has become an active research area in the recent past, as demonstrated by the abundance of manuscripts published in workshops and conferences during the last five years (i.e. The International Conference on Software Engineering, The International Software Process Workshop and The Hawaiian International Conference on System Sciences). Significant effort has been devoted to process modelling at the Software Engineering Institute as well (Kellner, 1987).

Process modelling describes the development process by:

- communicating the roles and responsibilities of both individuals and groups
- specifying where, within the software process, software products are created and changed
- depicting where measurement is performed
- supporting process improvement
- describing the communication channels between development groups

3.2. The process modelling technique

In theory, a new software process can be specified then simply implemented. In practice, a new software process cannot simply be defined then put into practice. We recognized that the existing process, no matter how chaotic or inefficient, was embedded in the organizational culture and could not be easily or rapidly changed. In order to specify a software process,

we carefully considered the software product, the existing process, the organization, the complexity of the software product and the past success of the product.

Changes to the existing process were based on an evolutionary approach. Once changes were proposed, the impact of each change was resolved, and the changes then incrementally implemented. Groups throughout the organization, (i.e. project management, software engineers, configuration management, quality assurance, testers, etc.) analysed the impact of changing their phase of the software process on the entire system development and maintenance process. The full impact of process change on these groups was considered and potential conflicts resolved. Changes implemented incrementally were monitored to ensure change was controlled and adverse affects recognized.

Process models are constructed using specific modelling methods or techniques. These techniques impart certain characteristics to the models they produce. Development of a modelling technique begins by specifying the characteristics desired in the resulting models.

The characteristics required by our models included:

- clear representation of the flow of control between and among process activities
- integration of functional, behavioural and organizational views within a single model
- representation of the process in multiple levels of implementation detail
- integration of process measurement
- documentation of the relationship between product and process

Application of the software engineering principles of information hiding, top-down functional decomposition, and stepwise refinement to process modelling imparts many desirable attributes, including modularity, strong cohesion and weak coupling to the models produced. In addition, representing the process in multiple levels of increasing detail makes the models usable by a wide variety of project personnel, from programmers to upper level managers.

Model construction must be carefully constrained to produce process models with the characteristics desired. Development of the modelling technique involved four major steps, each intended to impart some of the desired characteristics to the resulting models. First, it was determined that three tiers, each with increasing amounts of detail, be used to construct process models. Second, the purpose of each tier was defined. Third, model traceability constraints were specified to ensure consistent representation of a process between different tiers. Finally, rules governing the generation of models through all tiers, and specific rules for the generation of each tier were defined.

The three tiered technique used to create process models is shown in Figure 2, and described as follows:

1. Tier 1—Phase Tier. The purpose of Tier 1 is to name each phase within the process. Each phase in Tier 1 specifies the products input and output, interphase communication, measurements and the group responsible for implementing the majority of the phase.
2. Tier 2—Task Tier. The purpose of this tier is to describe *what* major tasks comprise a phase. A separate task tier exists for each phase.
3. Tier 3—Procedure Tier. The purpose of this tier is to specify *how* the major tasks of Tier 2 are performed. Tier 3 describes the major implementation steps for each task shown in Tier 2. A separate procedure tier exists for each phase.

The purpose of each tier is to elaborate progressively the activities within each phase. The

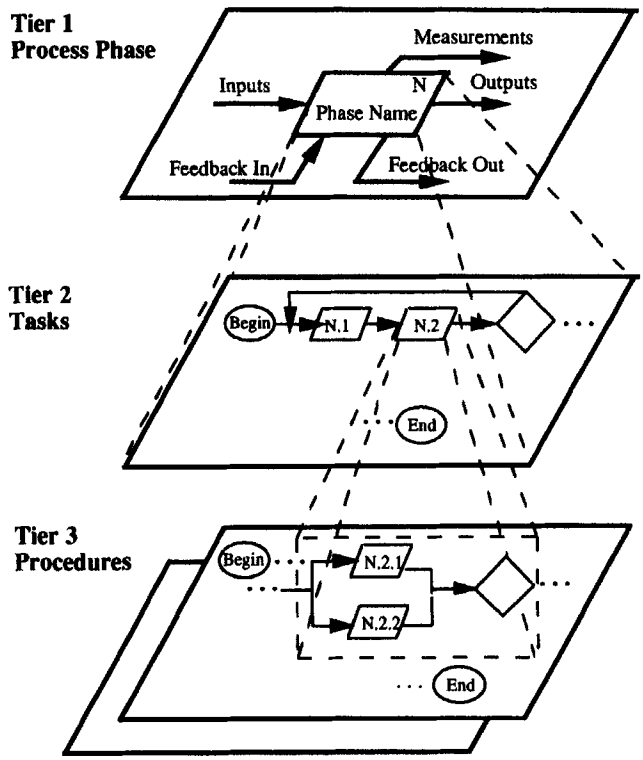

Figure 2. Modelling structure

tiers relate to each other in specific and well-documented ways. Sets of tiers, such as Tiers 1 and 2, or Tiers 2 and 3, form self-contained graphical representations of process phases.

The traceability constraints ensure each construct shown in a tier has a consistent, well-documented relationship to constructs on the next higher tier of abstraction as well as the next lower tier of detail. A structured numbering scheme ensures each task consists of a single set of procedures. Similarly, procedure steps unambiguously implement a single procedure. Additional traceability constraints and generation rules support tier to tier verification of the process representation.

A standard set of terms defines the information added to control flow diagrams. The terms and their definitions,shown in Figure 3, are as follows:

1. *Inputs*: products delivered at the start of a phase from a previous phase or from external sources. Inputs initiate the current phase.
2. *Outputs*: products generated by a phase that are inputs to a subsequent phase or are necessary documentation at Tier 1.
3. *Feedback In*: a change, constraint or other impact accepted by a phase generated by another phase at Tier 1.
4. *Feedback Out*: a change, constraint or other impact generated by a phase accepted by another phase at Tier 1.
5. *Measurements*: values obtained during a phase used to quantitatively evaluate the process, product or phase.

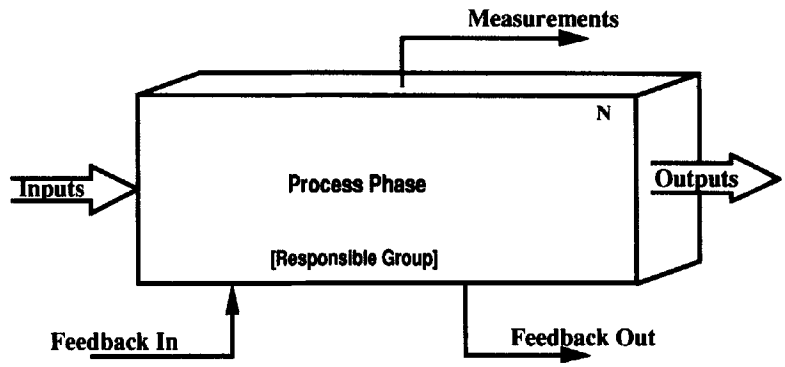

Figure 3. Tier 1 process phase

Figure 4 shows the symbology used in this modelling technique and illustrates the elegant simplicity of our approach to process description.

Eight generation rules govern construction of models throughout all tiers. These rules are:

1. All Outputs and Feedback Out shown on a Tier 1 phase block shall have clearly identified generation blocks on Tiers 2 and 3 of that phase.
2. All Inputs and Feedback In shown on a Tier 1 phase block shall have clearly identified blocks accepting the Input or Feedback In on Tiers 2 and 3 of that phase.
3. A block on any tier shall have exactly one control flow entrance point and one control flow exit point.
4. A decision block at any tier shall have exactly one control flow entrance point.
5. Blocks at all tiers shall be uniquely numbered. Tier 1 phases are numbered sequentially starting with 01. Blocks on Tiers 2 and 3 are numbered by adding a period and a number to the parent number of the block they describe. The following is an example of the numbering sequence for related blocks on Tiers 1 through 3.
   Tier 1: 01—phase 01
   Tier 2: 01.1, 01.2, 01.3, 01.4—the four tasks of phase 01
   Tier 3: 01.2.1, 01.2.2, 01.2.3—the three procedures implementing task 01.2
6. The single group within an organization responsible for a phase shall be stated in Tier 1. When the group performing a task or procedure is different from the group named in Tier 1, the description of the task or procedure step shall identify the responsible group.

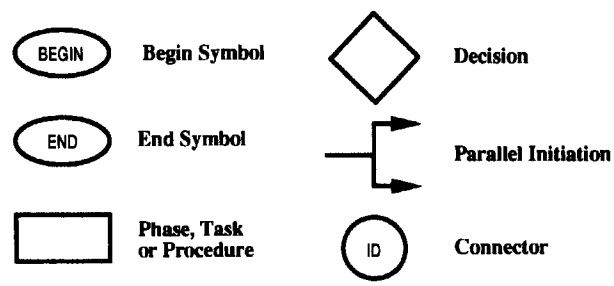

Figure 4. Process modelling symbology

7. Lines exiting a decision must go to a block or a different decision block.
8. All tiers shall start with a BEGIN symbol and terminate with an END symbol.

In addition to the general rules given above, specific generation rules for creating Tier 1 of a phase model are:

1. Tier 1 shall clearly depict each phase of the development process.
2. Tier 1 shall include for each phase the
   - Phase name and number,
   - Inputs,
   - Outputs,
   - Feedback In,
   - Feedback Out, and the
   - Responsible Group.
3. The phase is initiated upon receipt of the inputs.

The model generation rules for Tier 2 are:

1. Tier 2 shall provide an overview of the phase and should consist of four to ten tasks.
2. Each task shall perform a single function and this function shall not overlap with any other task.
3. Tier 2 is initiated upon receipt of the inputs shown in Tier 1.

Specific generation rules for Tier 3 are:

1. Tier 3 shall be a depiction of the procedures needed to implement the tasks in Tier 2 and maintains the same control flow shown in Tier 2.
2. Each procedure block shall map to a single task shown on Tier 2.
3. The single control flow entrance point and exit point of each task shown in Tier 2 shall be maintained in Tier 3.

3.3. Supporting process documentation

Process documentation supports and clarifies the graphic model. The documentation includes a written description of each procedure, entrance and exit criteria, applicable process and product standards, and measures.

Written descriptions expand on, and clarify, each procedure block of the graphic model. If a procedure block is labelled 'Form Test Team', the written description for this block clarifies how many members from each group are on the team, how members are chosen, and how the members are managed.

Entrance and exit criteria specify the minimum criteria for initiation and termination of procedures. A procedure named 'Conduct Engineering Tests' cannot begin until integrated software is installed on the target system and a current version of the software specification document is available. The procedure concludes when each individual requirement has been tested and the results recorded.

The Department of Defense (DOD) software development standards, contractually implemented by the organization, require identification of exactly which standards apply to

each procedure. For example, the 'Perform Code Review' task must be implemented according to the organizational process standard and must address adherence to the applicable coding and commenting standards. The anomalies detected must be documented using standard corrective action forms. These standards support a project interpretation of the applicable DOD standards.

3.4. Example

Consider the Engineering Test and Evaluation phase of the software maintenance process employed on this project at Martin Marietta. This phase tests each of the computer programs included in the radar-based weapons system against the governing requirements.

Figure 5 presents all three tiers of the graphic model of the Engineering Test and Evaluation phase. The Martin Marietta process actually contains twelve process phases but six are combined with others to simplify the presentation. Certain specifics are also left out for proprietary reasons. The model shown in Figure 5 illustrates the detail and thoroughness of the process model and the integration of measurement into the model.

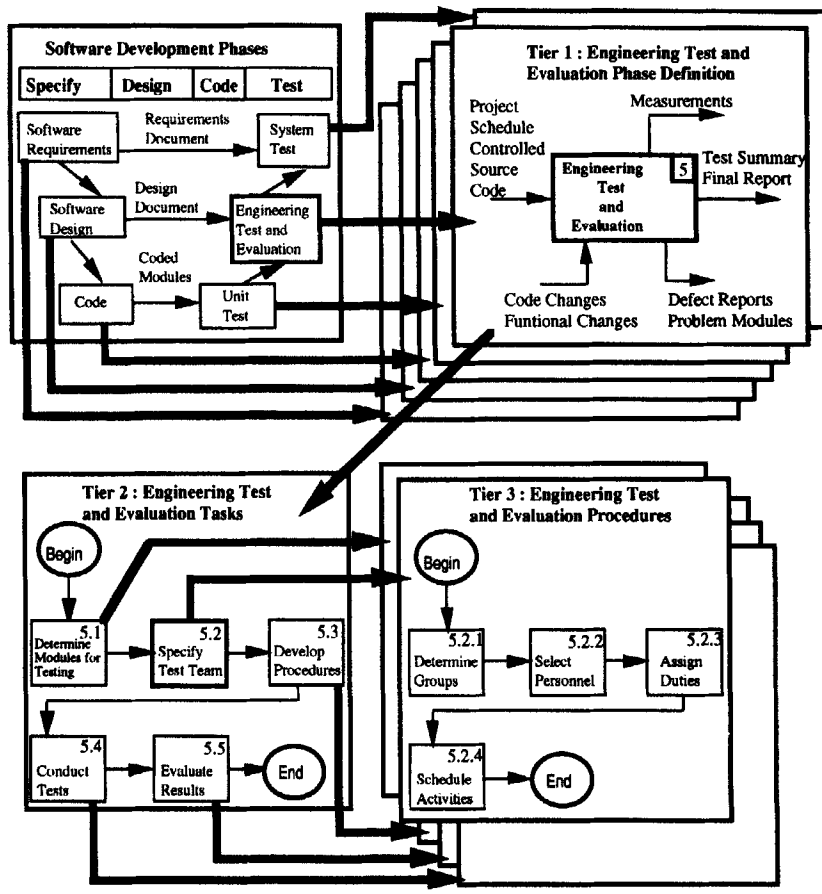

Figure 5. Process tiers

We found that defining process phases based on our contractually required military standards provided a common organization-wide starting point. Three level, graphical definitions provided all the information needed to understand and execute the process phases. The written descriptions of each process phase are very effective in supplementing the graphic representations with information needed to clarify and describe each task.

The approach taken at Martin Marietta required representatives from throughout the organization to agree on process definitions, thereby gaining a stable foundation to implement continuous process improvement without debate on definitions. It is too early to make a complete assessment of the success of this effort. However, analysis to date illustrates that continuous process improvement opportunities have been identified as an offshoot of process definition, with the organization becoming confident in the process modelling activities as an important way of representing business activities.

It should also be noted that a straightforward, easily understood and maintained method of representing processes was chosen with little regard to CASE related methodologies. Tools were then chosen which best supported both creation and maintenance of the modelling.

4. MEASURING THE SOFTWARE MAINTENANCE PROCESS

4.1. Measuring by level

The measurement-based software process includes the capture of software product and process data throughout the process. Software product measures track the progress and trace the impact of changes to individual software products, and quantify the impact of changes across software products. Process measures provide data by process phase and major activity within the process. Relationships between process and product measures are also measured.

We desired measures at three different levels, the baseline level (a high level view of all element computer programs as a single software system), the element level (a view of each element computer program), and the specific product and activity level. Baseline level measures consist of product and process measures of each element computer program in the system. Element level measures capture process and product data at the individual element level. Product and activity level measures are fine grain measures of both product and process within an element.

Measures at each level correspond with the three tiers in the process model and quantify process and product attributes from three different views. In addition, each increasingly detailed measurement level supports higher level measures in specific and important ways. For example, if baseline measures show an element is not meeting its budget criteria for a baseline, element level measures can provide additional information. The element level measures would indicate which upgrade items are troublesome and by the magnitude of the problem. The product and activity level measures could then highlight where additional effort was expended and what products required this effort.

At each level the measures focus on areas of particular interest. The areas of most interest at the baseline level include the factors influencing the quality, timeliness and cost of elements in the software system. For example, baseline measures capture the number of known defects in each element at delivery time, progress of the maintenance effort for each element, and the total cost as a percentage of budget. At the project level the measures focus on quality, effort, progress and status per software element. Element level measures capture the number

of defects detected during tests of each upgrade item, the actual effort as a percentage of the estimated effort to implement upgrade items, the progress of unit testing, and the resources allocated for each upgrade. The fine grain measures provide insight into modules comprising the system. For example, percentage of change, error-proneness, and McCabe's cylclomatic complexity per module are measured. Process measures provide feedback on the effectiveness, effort expended and progress for selected activities.

The measures selected include classic measures described by Pfleeger and Shultz, as well as basic product and process data (Pfleeger, 1993; Schultz, 1988). For example, the impact of change measured in the number of source lines of code (SLOCs) added, deleted and changed is recorded for each change to the computer programs.

4.2. Product measures

The maintenance process includes both deliverable and non-deliverable work products. Changes to these work products, and the status of each change, must be tracked and traced in order to effectively manage the maintenance process. The numerous types of work products, as well as the volume of work products generated during a baseline, require a tracking methodology supported by software tools. However, the strategy was to choose measures that were best for the processes and methods used by our organization.

Our measurement approach closely followed the process modelling approach and is shown in Figure 6. Measurements acquired at the module level provide the most detailed product view. Module level measures accumulate to element level measures, providing an evaluation of the entire element computer program. Element level measures integrate into system level measures reviewed at specific points in the maintenance process and at system demonstration which concludes a maintanence project.

Product measures at the baseline level focus on product characteristics with high visibility to the customer. These measures include the number of upgrades to be implemented, the number of defects detected, and the number of uncorrected errors along with their operational priority for the software system. The priority of each uncorrected error is of particular importance in real-time, embedded systems. Upper-level management is typically more concerned with process measures, but must be aware of product measures visible by, or of particular interest to, the customer.

Element level measures capture more detailed product data by individual element for a baseline effort. A maintenance effort begins with a contract, containing specification of the functional upgrades to be implemented in each element computer program. Each upgrade receives a unique title and acronym. As a maintenance project progresses, specification changes are written correcting or clarifying the upgrade items. Each specification change receives a unique number and is associated with the single upgrade item changed. Element level product measures then focus on each upgrade item and all associated specification changes. These measures include the impact of upgrade items and specification changes on the element computer program. Element managers are interested in product measures supporting baseline management within their elements, but are not as interested in more detailed product measures, illustrating the importance of a well-founded, multi-viewed software process and supporting measures.

The work product level of measurement involves detailed and specific product data. Upgrade items and specification changes approved for implementation are assigned a single distinct computer program change number. The computer program change number traces the

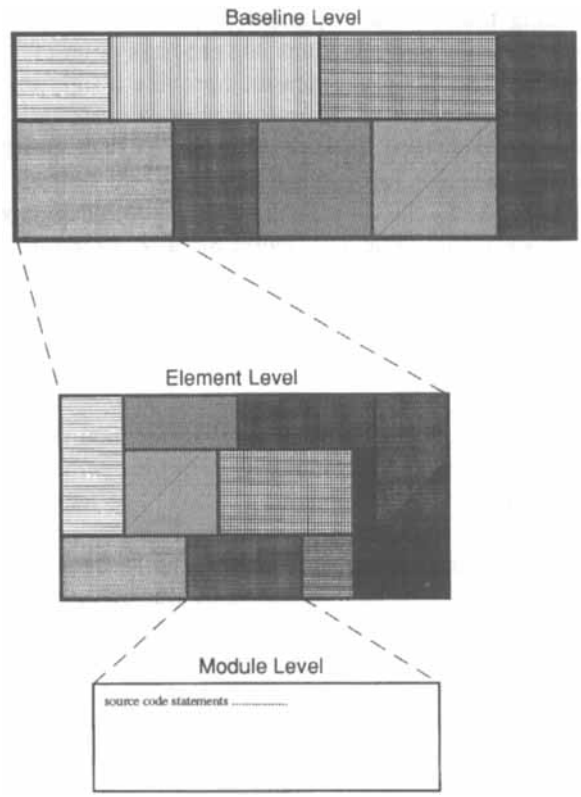

Figure 6. Three level product measurement

magnitude of the impact of implementing upgrade items and specification changes on modules comprising element computer programs. For each upgrade item and each specification change, both the estimate and the actual impact on each module in the system is recorded. The number of SLOCs (source lines of code) added, changed and deleted per module for each upgrade item and specification change are collected. Additional measures such as the change in module size, the percentage of the module altered, and the mean number of modules changed per specification change are computed from these basic measures.

These work product measures also track and trace error correction efforts. Each observed defect is assigned a unique defect number which is used to track subsequent review and correction activities. A review board determines if each defect is truly a defect and approves correction of a defect for a specific baseline. If a defect is approved for correction, a computer program change number is associated with the defect. The computer program change number is used to track and trace error correction impact on the product. Detailed product measures capture data by error correction and module within the element computer programs.

This most detailed level of measurement supports efforts to assess the quality of software experienced by the customer. Following delivery of a baseline, the customer assumes responsibility for installing the software system in a chosen ship or ship class, and maintaining the computer programs until delivery of the next baseline. The customer captures maintenance data about delivered computer programs including number and type of defects, and the impact of error correction on the computer programs. The single, consistent measurement approach

followed by both Martin Marietta and the customer allows detailed statistical analysis to be performed which can benefit both organizations.

The extensive numbering systems track the status and impact of changes to the element computer programs. Recording and maintaining data in association with these numbering schemes requires additional effort by already overburdened software professionals. Without these tracking and control systems, monitoring change status, tracing change impact, and analysing process–product relationships would be extremely difficult.

4.3. Process measures

Process measures are collected from the same three levels of detail used to collect product data. Process measures consistently focus on effort expended, process effectiveness and resource allocation throughout all three levels. The measures provide increasingly more detailed data about process tasks and activities as one moves down through the process tiers.

Baseline process measures include effort expended, progress toward baseline milestones and task completion rates per element. Effort expended per element provides insight into cost and schedule status, and provides feedback on the accuracy of initial effort estimates. Progress measures assess the status of each element within each major process phase. Task completion rates, used in conjunction with progress measures can be used to replan process phase completion dates. These process measures provide important information to upper level management without overwhelming them with detailed metric data.

Element level process measures include effort expended, progress, and task completion status, but focus on upgrade items, specification changes and error corrections within the element. For example, the effort expended to implement each upgrade item is captured. Progress is monitored by recording the number of requirements designed, implemented, integrated and tested per upgrade item. Task completion measures monitor the number of unit tests completed per week the number of defects corrected per week, and the specification changes written per month.

The most specific level of process measurement includes effort expended per activity, work product and progress. Effort expended measures capture data per activity such as unit test time and work per product, e.g., number of days spent correcting each software module for each error corrected. Progress measures quantify fine grain activities such as the total number of tests run per specification change.

The measurement approach results in the collection of a large amount of data related in many important ways. Measures are carefully collected, validated and stored in a relational database. The collection approach requires automation to reduce the impact of measurement on the maintenance staff. While the collection and storage of measures are extremely important, the procedures required to capture measures cannot overburden the maintenance team and prevent them from building high quality products on time and within budget.

4.4. Data collection and storage

Process measures for a large, complex project require the collection of large amounts of data by many groups and organizations participating in the maintenance effort. The process specifies measures by process phase, task and procedure as well as by many different deliverable and non-deliverable products. Our implementation of a sound measurement-based software process demanded well-defined and versatile tools to support its requirements. A

Sun wide-area network providing access to a central database, and Graphical User Interface (GUI) met the requirements of our process.

The design of the database supports storage of data collected from the various user groups and organizations involved in the process. Data supplied by multiple sources are integrated based on a third-normal form relational database schema. Data are stored by document, computer program, defect, error correction form and specification change as well as by process phase, task and procedure. The result is an overall conglomeration of data which, when accessed as a whole, are capable of presenting an accurate view of the entire process in multiple levels of detail.

In addition to a database design consistent with the defined process, maintaining the integrity of the data was critical to the accuracy and usefulness of the system. While the database prevents unauthorized access to the data, the GUI supports data integrity by controlling concurrent access.

The GUI strictly controls access to the data, preventing corruption and inconsistency, and guiding adherence to the specified process. Data are entered manually from within each organization. Distributed access to the database via the GUI presents the possibility that users may concurrently alter data. Therefore the GUI had to ensure that no inconsistencies resulted from their input actions by implementing a table locking scheme. In addition, data validation is also performed by the GUI prior to storing data in the database. The GUI also ensures that data are entered in the proper sequence by authorized personnel. Though each single user may only be permitted to enter a small portion of the data, the GUI integrates all data to permit a global view of the process.

For example, when a user enters a defect report or a specification change into the system, the GUI requires a minimum set of data necessary to support further review and processing. The groups and organizations impacted by a defect or specification change view these data and supply additional data required by the review boards. A defect report is acted upon by the Software Implementation Board (SIB) which reviews, evaluates, classifies and prioritizes the problem. A specification requirements review board considers specification changes in much the same manner. In either case, the GUI supports these reviews by storing status fields reflecting the action taken by the appropriate board. If the defect or specification change is approved, a computer program change number is created and linked to the defect or specification, thereby maintaining change initiation history. As implementation of the change progresses, field dependencies ensure that data entered follow the sequence specified by the process model. For example, a project manager must enter an approval status into a specific field to allow work to commence on a problem prior to a computer program developer entering status reflecting current stages of implementation.

Since one goal of the software process is to support the groups and organizations implementing the software process, all data entry was assumed to be performed using the GUI. There are, however, instances where groups or organizations generate data from source code using tools which are not applicable to other groups and organizations. Adhering to an evolutionary rather than revolutionary change approach, integration of this type of data is accomplished using tightly-controlled, automated cross-platform procedures. These procedures enforce the same data integrity rules implemented in the GUI. As a result, the data become an augmentation rather than an exception to the process. This benefits software process measures at all levels.

In addition to the data entry required of the users, and the automatic input from external sources, the system also automatically keeps track of when process milestones, indicated by

specific status settings in the database, are reached. For example, status settings would exist for development initiated, testing completed, problem fix delivery, etc. Consequently, the required organizational and automated inputs in concert with the automatically generated system data comprise the total data needed to accommodate all three levels of the software maintenance process measures. In this manner, the database acts as the reservoir of process information while the GUI and its supporting procedures act as a process administrator.

The measurement-based process must support all three levels of process measures. Because complex relationships exist between the data residing in the database, a plethora of data views are possible. For example, data can be viewed by computer program element, by upgrade item within an element, by specification change, by process phase, by module, by defect, or in many combinations of these keys. Figure 7 illustrates how the multiple organizations concurrently store and retrieve data as an integrated process component. These views provide important opportunities for process measure analysis, and hence process improvement.

4.5. Metric analysis

Metrics analysis techniques include graphical representations and statistical analysis. Bar charts, timelines and composite graphs are simple yet important techniques for analysing software metrics. The graphs produced highlight problem areas, indicate trends and convey status information to project management and staff. Statistical analysis supplements graphs, investigating in further detail relationships suggested by the graphs.

We sought analysis techniques firmly grounded in mathematics to assess product and process quality, predict future process and product characteristics, quantify the strength of associations between measures, and classify processes and products. The techniques described by Schneidewind were selected and put into practice (Schneidewind, 1992). The field of Management Science consistently applies statistical techniques to these problems with impressive results. The statistical techniques chosen for data analysis include

- regression
- linear correlation
- rank correlation
- contingency table analysis
- Mann–Whitney rank sum test

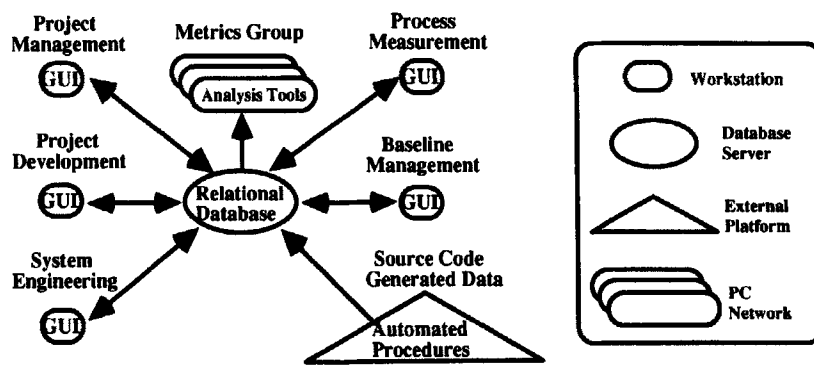

Figure 7. Status tracking and retrieval system

Linear and non-linear regression determines the ability to predict one variable from one or more other variables. Association is analysed using linear and rank correlation. Linear correlation measures the linear relationship between two variables. Rank correlation does not assume a linear relationship but rather tests the relationship based on ranks assigned to data values (the rank of a data value is the position of the data value in an ordered list of all data values). A simple classification of software products based on data available early in the maintenance process which predicts future product characteristics is measured using contingency tables. The Mann–Whitney rank sum test is used to compare the means of samples drawn from two or more independent populations to detect population differences (Ott, 1988).

Software tools

The software process, the software measures collected, existing maintenance practices, and the analysis techniques must be considered when selecting software tools to support data collection, storage, organization and analysis. In addition, budget constraints, available resources, existing platforms, required training and the impact of change on current practices also factor into the selection and integration of new tools in the software process. The software process requires tools to support informed management decision processes through:

- tracking and controlling specification changes
- recording and tracking defects
- recording, tracking and reporting the status of error correction activities
- linking product change to process activities
- collecting and storing software measures
- analysing metric data to support project management and process improvement priorities

A central database with distributed access via a graphical user interface (GUI), supported these needs. Data flow and validation are tightly controlled by the GUI to prevent possible corruption. Statistical analysis software is used to access the database, and provide many different types of charts, graphs and reports. This continuous improvement process keeps each element of the process under constant scrutiny.

The software metric analysis techniques include graphing and statistical analysis. Downloading data to personal computers allows users to generate graphs using a number of predefined graph generators or by constructing graphs of their own. Statistical analysis quantitatively evaluates relationships between and among process and product data.

Graphs

Graphs provide important metric information quickly and easily. Management and staff frequently need graphic representations of the metric data and are able to rapidly analyse the data when presented in graphic form. Users have access to a basic set of graphs and have the ability to generate many different types of graphs with a minimum of overhead. In addition, users have the ability to generate graphs not contained in the basic set. This is possible by downloading data from the database to graphical analysis software residing on the user PCs.

An Ethernet connection combined with commercial middleware provides workstation to

PC communication. A data retrieval software tool installed on the PCs grants widely distributed users access to the data. Users can easily download the data, then use graphical analysis software to create a myriad of graphical representations of the data.

Statistical analysis—first example

Statistical analysis of software measures quantitatively tests relationships between and among process and product measures. These relationships can be used to assess software product quality and process effectiveness, or to predict product or process characteristics early in the maintenance process. The two examples which follow demonstrate how these statistical techniques can be used.

The control program is composed of modules which are made up of procedures. Procedures consist of source code and comments. Computer program change numbers entered on each changed or added line of code allow modifications to be traced to, and accumulated by, procedure and module.

Test results showed some modules contained many errors and other modules contained very few. Further, the modules containing many errors were not consistently error-prone across baseline efforts. Clearly, some process activity or product characteristic influenced the distribution of errors across modules. If project management could determine the error-prone modules in the current baseline, those modules could be inspected or more rigorously tested or even redesigned and recoded.

The relationship between upgrade items, upgrade specification changes and errors corrected per module in the control program was investigated. Specifically, we analysed the impact of upgrade items and upgrade specification changes on the reliability of modules within the control program. Reliability in this context means the error-proneness of modules and was measured by errors corrected per module. If relationships existed, the relationships were to be quantified.

The number of errors corrected per module following integration testing was captured as well as the number of upgrade items and upgrade specification changes affecting each module. The percentage of the module changed (in SLOCs) and the number of person-days expended per module to implement upgrade specification changes were also captured.

Rank correlation was applied to determine the association between errors corrected per module and the impact of upgrade items per module. Impact on modules is measured by the number of upgrades affecting a module, the number of upgrade specification changes affecting a module, and the percentage of the module changed to implement upgrade specification changes. Impact on the maintenance process is measured by the number of person-days expended implementing upgrade specification changes per module. We found that a strong rank correlation existed between errors corrected per module and upgrade item impact, as shown by the rank correlation coefficients in Table I.

Table 1. Number of errors corrected versus maintenance impact

	Number of upgrades	Number of upgrade specification changes	Percentage of module changed	Number of person-days
Number of errors corrected	0.803	0.855	0.767	0.808

Table 2. Number of defects versus upgrade item impact

	≤ Median number of upgrade items impacting	> Median number of upgrade items impacting
≤ Median number of errors corrected	22	5
> Median number of errors corrected	2	26

Table 3. Number of defects versus upgrade specification change impact

	≤ Median number of upgrade specification changes impacting	> Median number of upgrade specification changes impacting
≤ Median number of errors corrected	25	3
> Median number of errors corrected	6	22

A simple classification of modules according to upgrade impact which predicts modules as having more or less than a specific number of errors was also desired. Contingency tables were also created using median values of errors corrected, number of upgrades affecting a module and the number of upgrade specification changes affecting a module. The contingency tables shown in Tables II and III, both have *P*-values of less than 0.005, and hence point to significant relationships.

The rank correlation coefficient shows significant correlation exists between errors corrected and upgrade item impact. Strong correlation suggests modules can be ranked using upgrade impact. This ranking predicts the ranking of modules based on errors corrected. Modules predicted to be error-prone can be selected for module inspection following the coding phase in an attempt to detect errors prior to lengthy and expensive test phases. Module ranking can also be used to select modules for additional unit and module integration testing prior to system testing.

Table III shows that selecting modules based on upgrade item impact greater than the median number modules selected with upgrade impact greater than the median value, 26 of them (84 per cent) actually had more than the median number of errors per module.

The analysis quantitatively evaluates the impact of upgrade items on modules. Modules can be rank ordered based on upgrade item impact or the historical median value of upgrade item impact can be used to classify modules as more or less error-prone. Upgrade impact information is currently being used within the organization to select modules for additional code walkthroughs or more intensive testing.

Statistical analysis—second example

The second example of statistical analysis focuses on revising the estimates of the effort required to implement upgrade items within each element computer program. The contract estimates for each upgrade item needing to be revised during the maintenance process in order to effectively manage the effort through delivery. As each baseline effort progresses,

data become available about the progress and status of the current effort. These data, used in conjunction with historical information, can provide valuable information about the current baseline effort. Accurate estimates of the effort required to implement upgrade items are very important for baseline planning and management.

One goal of the analysis of upgrade items was to determine if the actual effort required to implement an upgrade item could be predicted at two critical points in the maintenance process: immediately following PDR and immediately following CDR. If predictions could be made, the accuracy and timeliness must also be determined.

Applying the statistical technique of multiple linear regression, two predictive equations were derived, the first using data available at the PDR and the second using data available at the CDR. Table IV presents the results obtained using multiple regression for both of these sets of data. The data items included in the regression equations were chosen using stepwise regression techniques which select the most statistically significant independent variables from the set of available independent variables.

The prediction equations include three independent variables, two available at PDR and the third available at CDR. The estimate of person months in the contract and the number of specification changes approved at the PDR for each upgrade item are used in the PDR prediction equation. These same independent variables and the estimate of SLOCs needed to be changed to implement each upgrade are included in the CDR prediction equation. The dependent variable in both cases is the actual number of person-months needed to implement an upgrade item. The dependent variable equations were constructed for PDR and CDR respectively as follows:

Actual person-months = $-2.76 + 0.46 \times$ Contract estimate of person-months $+ 15.75 \times$ Number of PDR specification changes

Actual person-months = $-8.65 + 0.44 \times$ Contract estimate of person-months $+ 18.09 \times$ Number of PDR specification changes $+ 0.02 \times$ CDR estimate of changed SLOCs

Analysis of the prediction errors (residuals) was performed to determine if prediction intervals could be constructed. The residuals of each regression equation are normally distributed. The residuals of each equation were divided at the median predicted value. The variance of

Table 4. Regression analysis of actual of effort in person-months

Prediction time	Dependent variable	Independent variables	R^2	P-value
PDR	Actual person-months	o Contract estimate of person-months o Number of PDR specification changes	0.82	0.0011
CDR	Actual person-months	o Contract estimate of person-months o Number of PDR specification changes o CDR estimate of changed SLOCs	0.91	0.0001

residuals from above the mean and below the mean were found to be statistically equivalent for both equations. These assumptions are necessary to construct prediction intervals around the regression equations. Such prediction intervals have been constructed.

The ability to construct prediction intervals increases organizational confidence in the regression equations. Prediction intervals are in use at this time. Project management uses the intervals as a window of variability of actual upgrade item effort.

5. CONCLUSIONS AND LESSONS LEARNED

The intensive, continuous, long-term nature of this project provided a wealth of challenging problems, effective solutions and interesting conclusions. Many of these specific problems and the corresponding solutions are missing from the process improvement literature. For example, the Capability Maturity Model describes process maturity goals and the activities needed to accomplish these goals, but fails to specify how to implement the activities given organizational history and mission, schedule and budget constraints, and current hardware and software systems. We found these issues not only to be significant, but, in fact, critical in specifying and implementing a measurement-based software process.

The changes made by the PIC to the existing software process focussed on prioritized problems and significant challenges facing our organization. While many organizations struggle to satisfy SEI criteria, we concentrated on important problem areas regardless of the effect on our SEI maturity rating. The PIC did not allow the desire to achieve a specific process maturity level to take precedence over more pressing issues. We refused to allocate valuable resources to improve areas of the process mandated by others but instead applied these resources where we percieved the greatest potential benefit to the organization would be gained. Specifically, the most expensive phase of the software process, maintenance, received the bulk of our attention.

Cultural resistance was the largest problem we faced in defining and measuring the software maintenance process. Our solution was two-fold: first, involve software project managers and personnel in defining the process and specifying the measures, and secondly, adopt and adhere to an evolutionary approach to change. Involvement of personnel across the organization was critical in accurately defining the process, ensuring the process would be followed, and gaining cultural confidence in the process. An evolutionary, rather than a revolutionary approach, controlled the rate of change and made each change a permanent part of the organizational software process. In addition, precisely specifying when in the process to collect measurement data, how to collect the data, and what the measurement data actually quantified would have been impossible without the significant and valuable contribution made by software maintenance personnel at Martin Marietta.

The elaborate tracing and tracking system specified to support the software process definition allows us to gather detailed measures at three levels. We recognized early that a detailed tracking and tracing system, involving multiple numbering systems for linkage and driven by a software process, required distributed access and a custom software package. Without automation the tracing and tracking scheme would be completely unmanageable for software personnel. The automated system allows distributed access by multiple organizations throughout our multiple building site. The single, central database ensures strict integrity control over data. Analysis tools available, and those currently under development, provide a variety of views and analysis techniques from both project management and software engineer desktops.

Our measurements and analysis resulted in significant accomplishments. Project managers can now accurately revise the initial estimates of the effort required to implement upgrade items at two significant points in the maintenance process. The impact of different types of change is quantitatively known and strategies for managing these types of changes are in place. The effectiveness of process phases can be quantitatively assessed, such as the ability to detect defects early in the software process. The factors affecting post-delivery product quality have been identified and the effect quantified. The factors drive current process improvement efforts. Finally, quantifying impact of late specification changes on the software product and the maintenance process provides insight into the severity of a problem area we knew existed in advance. What we didn't know, but discovered through this project, included the magnitude of the impact and the characteristics driving the impact. In short, we learned about the strengths and weaknesses of our own processes.

This project produced, and continues to produce, valuable results in the commercial world. Known relationships between process and product have been quantified and new relationships discovered. Product evaluation and process assessment led to new questions, indicated where problems exist and what types of solutions are applicable, and highlighted the need for additional hardware and software support.

Process improvements have resulted from three distinct areas: process modelling, graph analysis and statistical analysis. Process modelling forced groups both within Martin Marietta and other organizations to specify the activities they perform. Modelling also resulted in a global view of the process, clarifying the roles and responsibilities of each group and organization. Graph analysis illustrated process status and trends. Statistical analysis quantified known relationships and discovered impacts and associations that pinpointed areas to improve.

Specifying a mature software process is far less difficult than implementing one. Understanding cultural resistance, defining the software process, specifying the measurement data, and providing hardware and software support, require real-world solutions. Such solutions typically synergize people, software, hardware and systems in order to effectively implement a mature, repeatable, measured software process. These solutions must also meet very real schedule, budget and personnel constraints. While no shortage of challenges exist, we remain convinced that a measurement-based software maintenance process is critically important to successfully compete in the software marketplace.

References

Humphrey, W. S. (1989) *Managing the Software Process*, Addison-Wesley, Reading, MA.

Kellner, M. I. (1987) 'Software process modelling: value and experience', SEI Technical Review, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Ott, L. (1988) *An Introduction to Statistical Methods and Data Analysis*, PWS-Kent, Boston, MA.

Pfleeger, S. L. (1993) 'Lessons learned in building a corporate metrics program', *IEEE Software,* **10**(5), 67–74.

Schneidewind, N. F. (1992) 'Methodology for validating software metrics', *IEEE Transactions on Software Engineering,* **18**(3), 410–422.

Schultz, H. P. (1988) 'Software management metrics', Technical Report ESD-TR-88–001, The Mitre Corp, Bedford, MA.

Authors' biographies:

Joel Henry is Assistant Professor of Computer Science at East Tennessee State University. His research interests include the software process, software metrics, software engineering methodologies, and the object-orientated paradigm. Henry received both B.S. and M.S. degrees in computer science from Montana State University, and the Ph.D. in computer science from Virginia Tech.

Robert M. Blasewitz is a Principal Project Specialist in Computer Program Development for Lockheed Martin at Moorestown NJ. During his nearly 25 year career, he has been involved with total quality management (TQM) implementation, program process definition and standards, metrics program definition and implementation, computer program standards, and process improvement methodology. Other responsibilities have included technology insertion, software environment definition and tools selection, and DOD standards implementation. Mr. Basewitz has authored technical papers, and is the co-author of a book on microcomputer systems, and of a book on TQM for software development.

David Kettinger is a Consultant with Integrated Systems Consulting Group, Inc., based in Wayne, PA. He has a B.S. in Computer Science from Drexel University and over six years of client/server software design and development experience. His projects include applications in the defence, engineering, financial and pharmaceutical industries. He is currently leading a project team in the development of a Document Management system utilizing Documentum and Visual Basic.